# Azure DocumentDB 中的分区和缩放

[Microsoft Azure DocumentDB](/services/documentdb/) 旨在帮助你实现快速、可预测的性能并且随着应用程序的增长无缝扩展。本文概述 DocumentDB 分区的工作原理，并且描述如何配置 DocumentDB 集合以有效地扩展应用程序。

阅读本文后，你将能够回答以下问题：

* Azure DocumentDB 中的分区原理是什么？
* 如何配置 DocumentDB 中的分区
* 什么是分区键，以及如何为应用程序选取适当的分区键？

## DocumentDB 中的分区

在 DocumentDB 中，可在任何规模以毫秒级响应时间对无模式 JSON 文档进行存储和查询。DocumentDB 提供了容器来存储名为 **collections** 的数据。集合是逻辑资源，并且可以跨一个或多个物理分区或服务器。DocumentDB 基于存储大小和集合设置的吞吐量确定分区数。DocumentDB 中的每个分区都具有固定大小的与之关联的 SSD 支持的存储，并且会复制分区以实现高可用性。分区完全由 Azure DocumentDB 进行管理，因此，你无需编写复杂的代码或亲自管理分区。DocumentDB 集合在存储和吞吐量方面是**几乎无限制**。

分区是对应用程序完全透明。DocumentDB 通过 REST API 对单一集合资源的调用来支持快速读取和写入、SQL 和 LINQ 查询、基于 JavaScript 的事务逻辑、一致性级别和精细的访问控制。该服务处理跨分区发配数据并将查询请求路由到正确的分区。

它的工作原理是什么？ 当在 DocumentDB 中创建集合时，你会注意到，存在你可以指定的**分区键属性**配置值。这是文档内 DocumentDB 可用的 JSON 属性（或路径），以在多个服务器或分区间分发数据。DocumentDB 将对分区键值进行哈希，并使用经过哈希运算的结果来确定将在其中存储 JSON 文档的分区。具有相同分区键的所有文档将存储在同一分区中。

例如，假设一个应用程序在 DocumentDB 中存储有关员工和部门的数据。让我们选择 "department" 作为分区键属性，以便按部门扩展数据。在 DocumentDB 中的每个文档必须包含必需的 "id" 属性，该属性对每个具有相同分区键值（如 "Marketing“）的文档必须是唯一的。存储在集合中的每个文档都必须具有分区键和 ID 的唯一组合，例如 { "Department": "Marketing", "id": "0001" }、{ "Department": "Marketing", "id": "0002" } 和 { "Department": "Sales", "id": "0001" }。换言之，分区键、ID 的复合属性是集合的主键。

### 分区键

选择分区键是设计时需要做的一项重要决定。选择的 JSON 属性名必须具有一系列的值，且有望均匀地分布访问模式。分区键指定为 JSON 路径，例如 /department 表示属性部门。

下表显示分区键定义和与每个对应的 JSON 值的示例。

<tbody>  
 <tr>  
 <td valign="top"><p><strong>分区键路径</strong></p></td>  
 <td valign="top"><p><strong>说明</strong></p></td>  
 </tr>  
 <tr>  
 <td valign="top"><p>/department</p></td>  
 <td valign="top"><p>对应 doc.department 的 JSON 值，其中 doc 指的是文档。</p></td>  
 </tr>  
 <tr>  
 <td valign="top"><p>/properties/name</p></td>  
 <td valign="top"><p>对应 doc.properties.name 的 JSON 值，其中 doc 指的是文档（嵌套属性）。</p></td>  
 </tr>  
 <tr>  
 <td valign="top"><p>/id</p></td>  
 <td valign="top"><p>对应 doc.id 的 JSON 值（ID 和分区键是相同属性）。</p></td>  
 </tr>  
 <tr>  
 <td valign="top"><p>/"department name"</p></td>  
 <td valign="top"><p>对应 doc["department name"] 的 JSON 值，其中 doc 指的是文档。</p></td>  
 </tr>  
</tbody>

[AZURE.NOTE] 分区键路径的语法类似于索引策略路径的路径规范，关键差别在于路径对应属性而不是值，即末尾没有通配符。例如，你会指定 /department /? 以在部门下为值编制索引，但指定 /department 作为分区键定义。分区键路径以隐式的方式进行索引编制，而且不能使用索引策略覆盖从索引中排除它。

让我们看看所选的分区键如何影响应用程序的性能。

### 分区和设置的吞吐量

DocumentDB 旨在提供可预测的性能。在创建集合时，你希望根据**每秒**[**请求单位**](documentdb-request-units.md) **(RU)** 来保留吞吐量。会为每个请求分配请求单位费用，该费用与系统资源（如操作使用的 CPU 和 IO）的数量成正比。读取 1 kb 会话一致性文档将使用 1 请求单位。读取为 1 RU 时不考虑存储的项数量或同时运行的并发请求数。较大的文档要求更高的请求单位，具体取决于大小。如果你知道实体大小及为应用程序提供支持需要的读取次数，则可以设置应用程序读取所需的那个吞吐量。

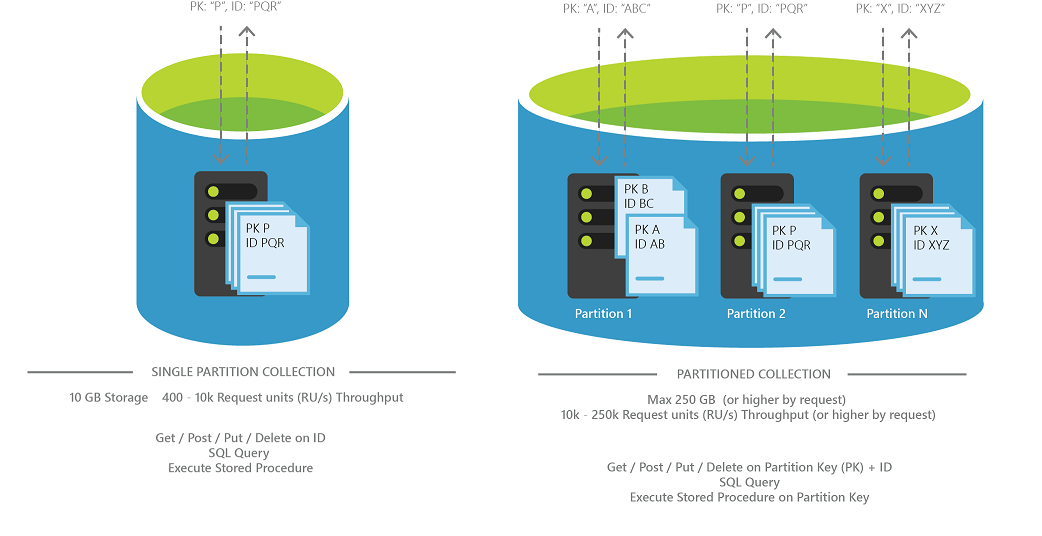
DocumentDB 存储文档时，它将基于分区键值在分区间均匀地分布它们。吞吐量也均匀分布在可用分区中，即每个分区的吞吐量 =（每个集合的总吞吐量）/（分区的数目）。

[AZURE.TIP] 为了实现集合的全部吞吐量，必须选择分区键，可用于在多个不同的分区键之间均匀分布请求。

## 单个分区和已分区的集合

DocumentDB 支持创建单个分区和已分区的集合。

* **已分区集合**可以跨越多个分区并支持数量非常大的存储和吞吐量。必须指定集合的分区键。
* **单个分区集合**具有较低价格选项和在所有集合数据之间进行查询和执行事务的能力。它们具有单个分区的可伸缩性和存储限制。无需指定这些集合的分区键。



DocumentDB 中的已分区集合

对于不需要大容量存储或大吞吐量的情形，单个分区集合非常适合。请注意，单个分区集合拥有单个分区的可扩展性和存储限制，即最多 10 GB 的存储和每秒多达 10,000 个的请求单位。

已分区集合可以支持数量非常大的存储和吞吐量。但是，默认的产品/服务将配置为最多 250 GB 的存储和增加到 250,000 个请求单位/秒。如果每个集合都需要更高的存储或吞吐量，请联系 [Azure 支持](/documentation/articles/documentdb-increase-limits)为你的帐户增加这些内容。

下表列出使用单个分区和已分区集合的区别：

<tbody>  
 <tr>  
 <td valign="top"><p></p></td>  
 <td valign="top"><p><strong>单个分区集合</strong></p></td>  
 <td valign="top"><p><strong>已分区集合</strong></p></td>  
 </tr>  
 <tr>  
 <td valign="top"><p>分区键</p></td>  
 <td valign="top"><p>无</p></td>  
 <td valign="top"><p>必选</p></td>  
 </tr>  
 <tr>  
 <td valign="top"><p>文档的主键</p></td>  
 <td valign="top"><p>"id"</p></td>  
 <td valign="top"><p>复合键 &lt;分区键> 和 "id"</p></td>  
 </tr>  
 <tr>  
 <td valign="top"><p>最小存储</p></td>  
 <td valign="top"><p>0 GB</p></td>  
 <td valign="top"><p>0 GB</p></td>  
 </tr>  
 <tr>  
 <td valign="top"><p>最大存储</p></td>  
 <td valign="top"><p>10 GB</p></td>  
 <td valign="top"><p>无限制（默认为 250 GB）</p></td>  
 </tr>  
 <tr>  
 <td valign="top"><p>最小吞吐量</p></td>  
 <td valign="top"><p>400 个请求单位/秒</p></td>  
 <td valign="top"><p>10,000 个请求单位/秒</p></td>  
 </tr>  
 <tr>  
 <td valign="top"><p>最大吞吐量</p></td>  
 <td valign="top"><p>10,000 个请求单位/秒</p></td>  
 <td valign="top"><p>无限制（默认为 250,000 个请求单位/秒）</p></td>  
 </tr>  
 <tr>  
 <td valign="top"><p>API 版本</p></td>  
 <td valign="top"><p>全部</p></td>  
 <td valign="top"><p>API 2015-12-16 及更新版本</p></td>  
 </tr>  
</tbody>

## 使用 SDK

Azure DocumentDB 增加了对 [REST API 版本 2015-12-16](https://msdn.microsoft.com/library/azure/dn781481.aspx) 的自动分区支持。为了创建已分区集合，必须在支持的 SDK 平台之一（.NET、Node.js、Java、Python）下载 SDK 版本 1.6.0 或更高版本。

### 创建已分区集合

下面的示例演示创建集合的 .NET 代码片段，以存储吞吐量为 20,000 个请求单位/秒的设备遥测数据。SDK 将设置 OfferThroughput 值（其反过来将设置 REST API 中的 x-ms-offer-throughput 请求标头）。这里，我们将 /deviceId 设为分区键。所选的分区键随集合元数据（如名称和索引策略）的其余部分一起保存。

对于此示例，我们选取了 deviceId，因为我们知道：(a) 由于存在大量的设备，写入可以跨分区均匀地分步并且我们可以扩展数据库以摄取海量数据，(b) 许多请求（如提取设备最近读取内容）仅限于单个设备 ID，并且可以从单个分区进行检索。

DocumentClient client = new DocumentClient(new Uri(endpoint), authKey);  
await client.CreateDatabaseAsync(new Database { Id = "db" });  
  
// Collection for device telemetry. Here the JSON property deviceId will be used as the partition key to   
// spread across partitions. Configured for 10K RU/s throughput and an indexing policy that supports   
// sorting against any number or string property.  
DocumentCollection myCollection = new DocumentCollection();  
myCollection.Id = "coll";  
myCollection.PartitionKey.Paths.Add("/deviceId");  
  
await client.CreateDocumentCollectionAsync(  
 UriFactory.CreateDatabaseUri("db"),  
 myCollection,  
 new RequestOptions { OfferThroughput = 20000 });

[AZURE.NOTE] 为了创建已分区集合，必须指定 > 10,000 个请求单位/秒的吞吐量值。由于吞吐量是 100 的倍数，因此必须是 10,100 或更多。

此方法可对 DocumentDB 调用 REST API，且该服务将基于所请求的吞吐量设置分区数。根据你的性能需求的发展，可以更改集合的吞吐量。有关详细信息，请参阅[性能级别](/documentation/articles/documentdb-performance-levels)。

### 读取和写入文档

现在，让我们将数据插入 DocumentDB。以下的示例类包含设备读取和对 CreateDocumentAsync 的调用，将新设备读数插入到集合。

public class DeviceReading  
{  
 [JsonProperty("id")]  
 public string Id;  
  
 [JsonProperty("deviceId")]  
 public string DeviceId;  
  
 [JsonConverter(typeof(IsoDateTimeConverter))]  
 [JsonProperty("readingTime")]  
 public DateTime ReadingTime;  
  
 [JsonProperty("metricType")]  
 public string MetricType;  
  
 [JsonProperty("unit")]  
 public string Unit;  
  
 [JsonProperty("metricValue")]  
 public double MetricValue;  
 }  
  
// Create a document. Here the partition key is extracted as "XMS-0001" based on the collection definition  
await client.CreateDocumentAsync(  
 UriFactory.CreateDocumentCollectionUri("db", "coll"),  
 new DeviceReading  
 {  
 Id = "XMS-001-FE24C",  
 DeviceId = "XMS-0001",  
 MetricType = "Temperature",  
 MetricValue = 105.00,  
 Unit = "Fahrenheit",  
 ReadingTime = DateTime.UtcNow  
 });

让我们来按分区键和 ID 读取文档，更新该文档，最后按分区键和 ID 将其删除。请注意，读取包括 PartitionKey 值（对应 REST API 中的 x-ms-documentdb-partitionkey 请求标头）。

// Read document. Needs the partition key and the ID to be specified  
Document result = await client.ReadDocumentAsync(  
 UriFactory.CreateDocumentUri("db", "coll", "XMS-001-FE24C"),   
 new RequestOptions { PartitionKey = new PartitionKey("XMS-0001") });  
  
DeviceReading reading = (DeviceReading)(dynamic)result;  
  
// Update the document. Partition key is not required, again extracted from the document  
reading.MetricValue = 104;  
reading.ReadingTime = DateTime.UtcNow;  
  
await client.ReplaceDocumentAsync(  
 UriFactory.CreateDocumentUri("db", "coll", "XMS-001-FE24C"),   
 reading);  
  
// Delete document. Needs partition key  
await client.DeleteDocumentAsync(  
 UriFactory.CreateDocumentUri("db", "coll", "XMS-001-FE24C"),   
 new RequestOptions { PartitionKey = new PartitionKey("XMS-0001") });

### 查询已分区集合

当在已分区集合中查询数据时，DocumentDB 会自动将查询路由到筛选器（如果有）中所指定分区键值对应的分区。例如，此查询将只路由到包含分区键“XMS-0001”的分区。

// Query using partition key  
IQueryable<DeviceReading> query = client.CreateDocumentQuery<DeviceReading>(  
 UriFactory.CreateDocumentCollectionUri("db", "coll"))  
 .Where(m => m.MetricType == "Temperature" && m.DeviceId == "XMS-0001");

下面的查询在分区键 (DeviceId) 上没有筛选器，并且以扇形展开到针对分区索引执行该查询的所有分区。请注意，必须指定 EnableCrossPartitionQuery（REST API 中的 x-ms-documentdb-query-enablecrosspartition）以使 SDK 跨分区执行查询。

// Query across partition keys  
IQueryable<DeviceReading> crossPartitionQuery = client.CreateDocumentQuery<DeviceReading>(  
 UriFactory.CreateDocumentCollectionUri("db", "coll"),   
 new FeedOptions { EnableCrossPartitionQuery = true })  
 .Where(m => m.MetricType == "Temperature" && m.MetricValue > 100);

### 执行存储过程

你还可以对具有相同设备 ID 的文档执行原子事务，例如，如果你要在单个文档中维护聚合或设备的最新状态。

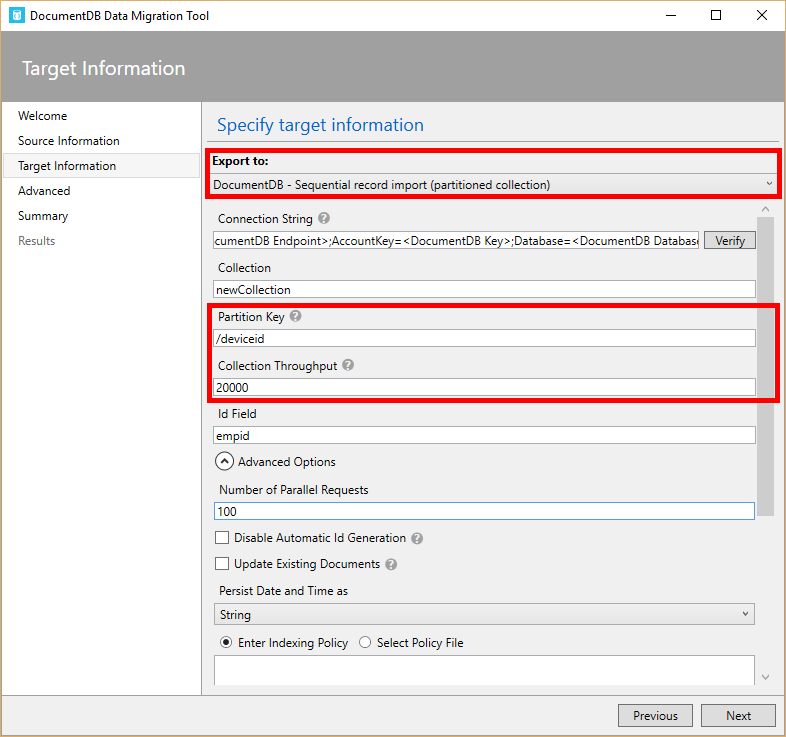
await client.ExecuteStoredProcedureAsync<DeviceReading>(  
 UriFactory.CreateStoredProcedureUri("db", "coll", "SetLatestStateAcrossReadings"),  
 "XMS-001-FE24C",  
 new RequestOptions { PartitionKey = new PartitionKey("XMS-001") });

在下一节，我们将介绍如何从单个分区集合移动到已分区集合。

### 从单个分区集合迁移到已分区集合 当使用单个分区集合的应用程序需要更高的吞吐量 (>10,000 RU/s) 或更大的数据存储 (>10GB) 时，可以使用 [DocumentDB 数据迁移工具](http://www.microsoft.com/downloads/details.aspx?FamilyID=cda7703a-2774-4c07-adcc-ad02ddc1a44d)将单个分区集合中的数据迁移到已分区集合。

从单个分区集合迁移到已分区集合

1. 将单个分区集合中的数据导出到 JSON。有关其他信息，请参阅[导出到 JSON 文件](/documentation/articles/documentdb-import-data#export-to-json-file)。
2. 将数据导入到使用分区键定义创建的、吞吐量超过 10,000 个请求单位/秒的已分区集合，如下例所示。有关其他信息，请参阅[导入到 DocumentDB](/documentation/articles/documentdb-import-data#DocumentDBSeqTarget)。



将数据迁移到 DocumentDB 中的已分区集合

[AZURE.TIP] 为了实现更快的导入时间，请考虑将并行请求数增加到 100 或更多，从而充分利用已分区集合可用的更高吞吐量。

现在我们已经学完了基础知识，让我们看看当在 DocumentDB 中使用分区键时几个重要的设计注意事项。

## 设计分区

选择分区键是设计时需要做的一项重要决定。本节将介绍在为集合选择分区键时所涉及的一些利弊。

### 作为事务边界的分区键

所选的分区键应该权衡启用事务的需求与将实体分布到多个分区键（以确保可缩放的解决方案）的需求。一种极端情况，你可以为所有文档设置相同的分区键，但这可能会限制解决方案的可扩展性。另一种极端，你可以为每个文档指定唯一的分区键，这将实现高度可扩展性，但会阻止你通过存储过程和触发器跨文档事务进行使用。理想的分区键是这样的，你可以使用高效查询，并具有足够多基数以确保你的解决方案是可缩放的。

### 避免存储和性能瓶颈

还很重要的是选取一个属性可以允许在大量相异值间分布写入。对相同分区键的请求不能超过单个分区的吞吐量，否则会受到限制。因此选取不会导致应用程序中**“热点”**的分区键很重要。具有相同分区键的文档的总存储大小也不能超过 10 GB。

### 适当的分区键的示例

以下是有关如何选择应用程序的分区键的几个示例：

* 如果你要实现用户配置文件后端，则用户 ID 是分区键的一个不错选择。
* 如果你要存储 IoT 数据（如设备的状态），则设备 ID 是分区键的一个不错选择。
* 如果你正在使用 DocumentDB 记录时间序列数据，则主机名或进程 ID 是分区键的一个不错选择。
* 如果你拥有多租户体系结构，则租户 ID 是分区键的一个不错选择。

请注意，在一些用例中（如上面所述的 IoT 和用户配置文件），分区键可能与 ID（文档键）相同。在其他用例（如时间序列数据）中，可能拥有与该 ID 不同的分区键。

### 分区和记录/时间序列数据

DocumentDB 最常见的使用案例之一是记录和遥测。选取适当的分区键是很重要的，因为你可能需要读取/写入大量数据。选择将取决于读取和写入的速率以及你希望运行的查询种类。以下是有关如何选择适当分区键的一些提示。

* 如果你的用例涉及很长时间积累的小速率写入，并且需要按时间戳范围和其他筛选器进行查询，则使用时间戳（例如数据）汇总作为分区键是个好方法。这使你能够查询某日单个分区中的所有数据。
* 如果你的工作负荷是更常见的写入密集型，应使用不基于时间戳的分区键，以使 DocumentDB 可以跨多个分区均匀地分布写入。此处，主机名、进程 ID、活动 ID 或其他具有较大基数的属性是不错的选择。
* 第三种方法是混合型分区键，其中你有多个集合，一个用于每日/月，且分区键是类似主机名的粒度属性。这样做的好处是可以基于时间窗口设置不同性能级别，例如，当月的集合设置为更高的吞吐量，因为它维护读取和写入操作，而之前的月份吞吐量设置为较低，因为它们只维护读取。

### 分区和多租户

如果你要使用 DocumentDB 实现多租户应用程序，有两种模式来使用 DocumentDB 实现租户，一种是一个租户一个分区键，另一种是一个租户一个集合。下面是每种模式的优缺点：

* 一个租户一个分区键：在此模型中，租户共置于单个集合。但是，可以针对单个分区执行单个租户内的文档查询和插入。你可以在租户中的所有文档间实现事务逻辑。由于多个租户共享一个集合，你可以通过在单个集合内集中租户资源而不是为每个租户配置额外的多余空间来节约存储和吞吐量成本。缺点是你没有依据租户隔离性能。性能/吞吐量的增加适用于整个集合，针对性增加适用于租户。
* 一个租户一个集合：每个租户都有它自己的集合。在此模型中，你可以依据租户保留性能。因为 DocumentDB 新的基于消耗的定价模型，这种模型对拥有少量租户的多租户应用程序更加划算。

此外可以使用一种组合/分层的方法，该方法共置小租户并将较大的租户迁移至它们自己的集合中。

## 后续步骤

在本文中，我们已经介绍了分区在 Azure DocumentDB 中的工作原理，如何创建已分区的集合和如何为应用程序选取适当的分区键。

* 使用 [SDK](/documentation/articles/documentdb-sdk-dotnet) 或 [REST API](https://msdn.microsoft.com/library/azure/dn781481.aspx) 的编码入门
* 了解 [DocumentDB 中设置的吞吐量](/documentation/articles/documentdb-performance-levels)
* 如果你想要自定义应用程序执行分区的方式，可以插入自己的客户端分区实现。请参阅[客户端分区支持](/documentation/articles/documentdb-sharding)。